

Skewless Network Clock Synchronization

Enrique Mallada*, Xiaoqiao Meng[†], Michel Hack[†], Li Zhang[†], and Ao Tang*

* School of ECE, Cornell University, Ithaca, NY 14853, USA.

[†] IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA.

Email: {em464@,atang@ece.}cornell.edu, {xmeng,hack,zhangli}@us.ibm.com

Abstract—This paper examines synchronization of computer clocks connected via a data network and proposes a skewless algorithm to synchronize them. Unlike current solutions, which either estimate and compensate the frequency difference (skew) among clocks or introduce offset corrections that can generate jitter and possibly even backward jumps, our algorithm achieves synchronization without these problems. We first analyze the convergence property of the algorithm and provide necessary and sufficient conditions on the parameters to guarantee synchronization. We then implement our solution on a cluster of IBM BladeCenter servers running Linux and experimentally study its performance. In particular, through both analytical and experimental results, we show that our algorithm can converge even in the presence of timing loops. This marks a clear contrast with current standards such as NTP and PTP, where timing loops are specifically avoided. Furthermore, timing loops can even be beneficial in our scheme. For example, it is demonstrated that highly connected subnetworks can collectively outperform individual clients when the time source has large jitter.

I. INTRODUCTION

Keeping consistent time among different nodes in a network is a basic requirement of many distributed applications. Their internal clocks are usually poor and tend to drift apart from each other, generating inconsistent time values. Network clock synchronization refers to the ability of these devices to correct their clocks to match a global reference of time, such as the Universal Coordinated Time (UTC), by performing time measurements through the network. For example, for the Internet, network clock synchronization has been a subject of research and several different protocols have been proposed [1]–[6].

These protocols are used in a wide variety of applications with diverse precision requirements. For example, the current standard for IP networks, NTP [1], provides a precision range that varies from $100\mu\text{s}$ to a few milliseconds depending on the network environment. This is suitable for personal computers or applications that involve human interactions. On the other hand, the Precision Time Protocol (PTP) [2] and IBM Coordinated Cluster Time (CCT) solution [7] are able to achieve precision of the order of $1\mu\text{s}$ which is suitable for distributed applications that require strong time consistency such as special-purpose industrial automation, network measurements and real-time financial transactions.

There are two major difficulties that make the network clock synchronization problem challenging. First, the frequency of hardware clocks is sensitive to temperature and constantly varies. Second, the latency introduced by OS and network congestion delays result in errors in the time measurements.

Thus, most protocols introduce different ways of estimating the frequency mismatch (skew) [8], [9] and measuring the time difference (offset) [10], [11]. In particular, the extensive literature on skew estimation [9], [12]–[14] for clock synchronization suggests that accurate clock synchronization can only be achieved if a precise estimation of the skew is obtained.

This paper shows that focusing on skew estimation could be misleading. We provide a simple algorithm that is able to compensate the clock skew without any explicit estimation of it. Our algorithm only uses current offset information and an exponential average of the past offsets. Thus, it neither needs to store long offset history nor perform involved operations over them. We analyze the convergence of the algorithm and provide necessary and sufficient conditions for synchronization. The parameter values that guarantee synchronization depend on the interconnection topology, but there exists a subset of them that is independent of it and therefore of great practical interest.

We also discover another rather surprising fact. A common practice in the clock synchronization community is to avoid timing loops in the network [1, p. 3] [2, p. 16, s. 6.2]. This is because it is thought that timing loops can induce instability as stated in [1]: “*Drawing from the experience of the telephone industry, which learned such lessons at considerable cost, the subnet topology... must never be allowed to form a loop.*” Even though for some parameter values loops can produce instability, we show in this paper that a proper selection of them can guarantee convergence even in the presence of loops. Furthermore, we experimentally demonstrate in Section V that high connectivity between clients can actually help reduce the jitter of the synchronization error!

The rest of the paper is organized as follows. In Section II we describe how clocks are actually implemented in computers and how different protocols discipline them. Section III motivates and describes our algorithm together with an intuitive explanation of why it works. In Section IV, we analyze the algorithm and determine the set of parameter values and connectivity schemes under which synchronization is guaranteed. Experimental results evaluating the performance of the algorithm are presented in Section V. We conclude in Section VI.

II. COMPUTER CLOCKS AND SYNCHRONIZATION

Most computer architectures keep track of time using a register that is periodically increased by either hardware or kernel’s interrupt service routines (ISRs). On Linux platforms,

there are usually several different clock devices that can be selected as the clock source by changing the *clocksource* kernel parameter.

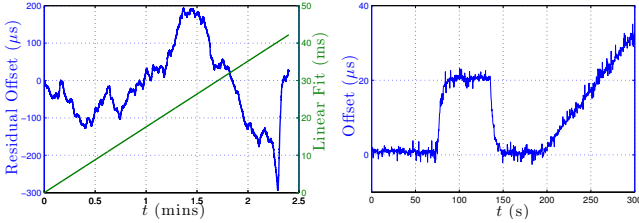
One particular counter that has recently been used by several clock synchronization protocols [4], [7] is the Time Stamp Counter (TSC) that counts the number of CPU cycles since the last restart. The TSC is a 64-bit counter that has a nominal increment period of δ^o . For example, in the IBM BladeCenter LS21 server, the CPU has a nominal frequency $f^o = 2399.711\text{MHz}$ which makes $\delta^o = 0.416\text{ns}$.

Using the TSC counter, the server can compute its own estimate $x(t)$ of the reference time t using the following equation

$$x(t) = \delta^o TSC(t) + x^o \quad (1)$$

where $TSC(t)$ is the counter's value at time t and x^o is the estimate of the time when the server was turned on (t^o). The value of the counter can be represented by $TSC(t) = \lfloor \frac{t-t^o}{\delta} \rfloor$ where δ is the actual CPU cycle period and $\lfloor \cdot \rfloor$ is the floor operator.

Since $\delta^o = 0.426\text{ns} \ll 1$, $\delta^o TSC(t)$ is usually approximated by $\delta^o \lfloor \frac{t-t^o}{\delta} \rfloor \approx r(t-t^o)$. $r := \frac{\delta^o}{\delta}$ represents the skew of the local clock with respect to its nominal value. When $r > 1$ ($r < 1$) the clock is running with frequency higher (lower) than $f^o = \frac{1}{\delta^o}$.



(a) Offset between two TSC counters (b) Effect of *adjtimex()* on linux time
Fig. 1: Comparison between two TSC counters and execution of *adjtimex* command

In an ideal scenario, r equals 1. Unfortunately, the skew r varies due to several factors including room temperature, among others. This is shown in Figure 1a where we plot the offset variations between the TSC counters of serv0 and serv1 in our testbed (Figure 2) over more than two days. Therefore, it is desirable to compensate the frequency error and compute $x(t)$ by introducing a skew correction s in (1).

Thus, $x(t)$ can be expressed as a linear function of the time

$$x(t) = rs(t-t^o) + x^o. \quad (2)$$

This linear map shows explicitly what are the two fundamental unknowns in a clock synchronization problem (t^o and r) and the two parameters that can be used to steer the clock (s and x^o).

To discipline $x(t)$ towards t one needs to estimate the offset $D^x(t) = t - x(t)$ at time t^o and the relative frequency error $f^{err} = \frac{1-r}{r}$. In fact, if these values were available at startup

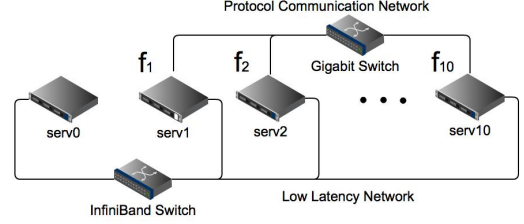


Fig. 2: Testbed of IBM BladeCenter blade servers

(something that in practice is not true), then one could just set $s = 1 + f^{err}$ and add an additional offset to (2) to get

$$\begin{aligned} x(t) &= r \left(1 + \frac{1-r}{r} \right) (t-t^o) + x^o + D^x(t^o) \\ &= 1(t-t^o) + x^o + (t^o - x^o) = t. \end{aligned}$$

Unfortunately, these values are in general unknown and variable. Thus, $f^{err}(t)$ and $D^x(t)$ need to be repeatedly estimated while the server is running, which introduces several constraints on how the clock can be disciplined as it may affect the execution of time sensitive applications.

To understand the differences between current protocols, we first rewrite the evolution of $x(t)$ based only on the time instants t_k in which the clock corrections are performed. We allow the skew correction s to vary over time, i.e. $s(t_k)$, and write $x(t_{k+1})$ as a function of $x(t_k)$. Thus, we obtain

$$x(t_{k+1}) = x(t_k) + \tau_k r s(t_k) + u^x(t_k) \quad (3)$$

$$s(t_{k+1}) = s(t_k) + u^s(t_k) \quad (4)$$

where $\tau_k = t_{k+1} - t_k$ is the time elapsed between adaptations; also known as poll interval [1].

The values $u^x(t_k)$ and $u^s(t_k)$ represent two different types of correction that a given protocol chooses to do at time t_k and are usually implemented within the interval (t_k, t_{k+1}) . $u^x(t_k)$ is usually referred to as *offset correction* and $u^s(t_k)$ as *skew correction*.

These corrections can be done in Linux OS using the *adjtimex()* interface. The commands

adjtimex -o offset and *adjtimex -f freq*,

where *offset* is in nanoseconds¹ (ns) and *freq* = 65536 equals 1ppm (parts per million), are equivalent to setting

$$u^x(t_k) = \text{offset} \cdot 1e^{-9}\text{s} \text{ and } u^s(t_k) = (1 + (\text{freq}/65536)) \cdot 1e^{-6}.$$

Figure 1b shows the execution of two offset corrections of $+20\mu\text{s}$ and $-20\mu\text{s}$, and one frequency correction of approx 0.3ppm. We used *offset* = ± 20000 and *freq* = 20000.

Some protocols prefer instead to maintain their own virtual version of $x(t)$ as for example IBM CCT [7] and RAD-clock [4]. This gives more control on how the corrections are implemented since it does not depend on kernel's routines.

¹In some implementations *adjtimex -o* also changes the frequency of the linux time. In such cases one can perform only offset corrections by immediately executing *adjtimex -f* to correct this change

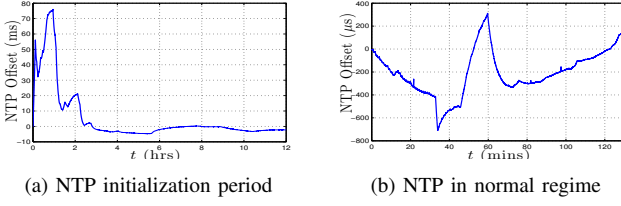


Fig. 3: Variations of NTP time using TSC as reference

We now proceed to summarize the different types of adaptations implemented by current protocols. The main differences between them are whether they use offset corrections, skew corrections, or both, and whether they update using offset values $D^x(t_k) = t_k - x(t_k)$, frequency errors $f^{err}(t_k) = \frac{1-rs(t_k)}{r}$, or both.

A. Offset corrections

This correction consists in using $u_s(t_k) = 0$ and either

$$u^x(t_k) = \kappa_1 D^x(t_k), \text{ or} \quad (5)$$

$$u^x(t_k) = \kappa_1 D^x(t_k) + \kappa_2 f^{err}(t_k), \quad (6)$$

where $\kappa_1, \kappa_2 > 0$. These adaptations are used by NTPv3 [15] and NTPv4 [1] respectively under ordinary conditions.

The protocols that use (5) or (6) have in general a slow initialization period as shown in Figure 3a. This is because the algorithm must first obtain a very accurate estimate of the initial frequency error $f^{err}(t^0)$ and set $s(t^0) = 1 + f^{err}(t^0)$. Furthermore, these updates usually generate non-smooth time evolutions as in Figure 3b and should be done carefully since they might introduce backward jumps ($x(t_{k+1}) < x(t_k)$).

B. Skew corrections

Another alternative that avoids using steep changes in time was proposed in [7]. This alternative does not introduce any offset correction, i.e. $u_x(t_k) = 0$, and updates the skew $s(t_k)$ using

$$u^s(t_k) = \kappa_1 D^x(t_k) + \kappa_2 f^{err}(t_k) \quad (7)$$

In [16] it was shown for a slightly modified version of (7) (used $r f^{err}(t_k)$ instead of $f^{err}(t_k)$) that under certain conditions in the parameter values, the algorithm achieves synchronization for very diverse network architectures.

However, the estimation of $f^{err}(t_k)$ is nontrivial as it is constantly changing with subsequent updates of $s(t_k)$ and it usually involves sophisticated computations [8], [9].

C. Skew and offset corrections

This type of correction allows dependence on only offset information $D^x(t_k)$ as input to $u^x(t_k)$ and $u^s(t_k)$. For instance, in [5] the update

$$u_x(t_k) = \kappa_1 D^x(t_k) \quad \text{and} \quad u_s(t_k) = \kappa_2 D^x(t_k) \quad (8)$$

as proposed.

This option allows the system to achieve synchronization without any skew estimation. But the cost of achieving it is introducing offset corrections in $x(t)$, incurring in the same disadvantages discussed in II-A

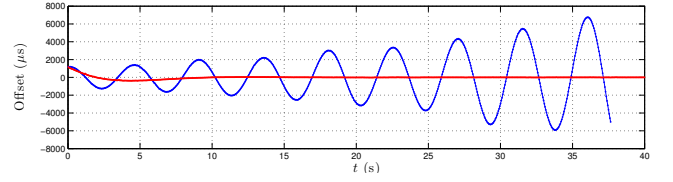


Fig. 4: Unstable clock steering using only offset information (9) and stable clock steering based on exponential average compensation(11)

III. SKEWLESS NETWORK SYNCHRONIZATION

We now present an algorithm that overcomes the limitations of the solutions described in Section II. In other words, our solution has the following two properties:

- 1) Smoothness: The protocol does not introduce steep changes on the time value, i.e. $u^x(t_k) \equiv 0$.
- 2) Skew independence: The protocol does not use skew information f^{err} as input.

After describing and motivating our algorithm, we show how the updating rule can be implemented in the context of a network environment.

The motivation behind the proposed solution comes from trying to compensate the problem that arises when one tries to naively impose properties 1) and 2), i.e. using

$$u^x(t_k) = 0 \quad \text{and} \quad u^s(t_k) = \kappa D^x(t_k). \quad (9)$$

Figure 4 shows that this type of clock corrections is unstable; the offset $D^x(t_k)$ of the slave clock oscillates with an exponentially increasing amplitude.

The oscillations in Figure 4 arise due to the fundamental limitations of using offset to update frequency. On the other hand, the exponential increase appears since at time t_{k+1} the update is based on the offset at time t_k . Right before updating (at t_{k+1}^-) the actual value of the offset has a correction

$$D^x(t_{k+1}^-) = D^x(t_k) + \tau_k r f^{err}(t_k)$$

which after noticing that $f^{err}(t_k) = f^{err}(t_{k+1}^-)$ amounts to an effective correction given by

$$u^s(t_k) = \kappa D^x(t_k) = \kappa D^x(t_{k+1}^-) - \kappa \tau_k r f^{err}(t_{k+1}^-).$$

Thus, at the moment of the correction the offset used implicitly includes a negative term in the frequency error that hurts synchronization. This is clearly seen in the case of a slower slave clock $f^{err}(t_{k+1}^-) = \frac{1-rs(t_{k+1}^-)}{r} > 0$ with a positive offset $D^x(t_{k+1}^-) = t_{k+1}^- - x(t_{k+1}^-) > 0$. While the first term of the correction tends to speed up the clock, the second term tends to slow it down.

One way to try to damp these unstable oscillations is to add a term that opposes the frequency error term. This is done in (7) by making $\kappa_2 > \kappa_1 \tau_k r$. However, there are other ways to generate such a term. For instance, consider the exponentially weighted moving average of the offset

$$y(t_{k+1}) = p D^x(t_k) + (1-p)y(t_k). \quad (10)$$

and update $s(t_k)$ using

$$u^s(t_k) = (\kappa + \gamma)D^x(t_k) - \gamma y(t_k).$$

If we again reference these values at the moment right before the correction (t_{k+1}^-) we have

$$u^s(t_k) = \kappa D^x(t_{k+1}^-) - (\kappa + \gamma)\tau_k r f^{err}(t_{k+1}^-) + \gamma(D^x(t_{k+1}^-) - y(t_{k+1}^-)).$$

So now, in the same situation as before ($f^{err}(t_{k+1}^-) > 0$ and $D^x(t_{k+1}^-) > 0$), we have a new term $\gamma(D^x(t_{k+1}^-) - y(t_{k+1}^-))$. This will be in general positive since the offset tends to further increase when the slave clock is slower and thus $D^x(t_{k+1}^-) - y(t_{k+1}^-) > 0$.

Thus, motivated by this discussion, we propose the following updating strategy:

$$u^x(t_k) = 0 \quad \text{and} \quad u^s(t_k) = \kappa_1 D^x(t_k) - \kappa_2 y(t_k) \quad (11)$$

where $\kappa_1 = \kappa$ and $\kappa_2 = \kappa + \gamma$. Figure 4 shows how the proposed strategy is able to compensate the oscillations without needing to estimate the value of $f^{err}(t_k)$. The stability of the algorithm will depend on how κ_1 , κ_2 and p are chosen. A detailed specification of these values is given in Section IV-B.

Finally, since we are interested in studying the effect of timing loops, we move away from the client-server configuration implicitly assumed in Section II and allow mutual or cyclic interactions among nodes. Each node i has its own clock with skew r_i and maintains its own values of $x_i(t_k)$, $s_i(t_k)$ and $y_i(t_k)$. The interactions between different nodes is described by a graph $G(V, E)$, where V represents the set of nodes ($i \in V$) and E the set of *directed* edges ij ; $ij \in E$ means node i can measure its offset with respect to j , $D_{ij}^x(t_k) = x_j(t_k) - x_i(t_k)$.

Within this context, a natural extension of (11) is to substitute $D^x(t_k)$ with the average of i 's neighbors offsets. Thus, we propose

$$s_i(t_{k+1}) = s_i(t_k) + \kappa_1 \frac{c}{d_i} \sum_{j \in \mathcal{N}_i} D_{ij}^x(t_k) - \kappa_2 y_i(t_k) \quad (12a)$$

$$y_i(t_{k+1}) = p \frac{c}{d_i} \sum_{j \in \mathcal{N}_i} D_{ij}^x(t_k) + (1 - p)y_i(t_k) \quad (12b)$$

where \mathcal{N}_i represents the set of neighbors of i and $d_i = |\mathcal{N}_i|$. The scalar c is a commit or gain factor that controls the offset's influence on the system.

Under this framework, many servers can affect the final frequency of the system. Thus in general, when the system synchronizes we obtain

$$x_i(t_k) = r^* t_k + x^* \quad i \in V. \quad (13)$$

The final frequency r^* and offset x^* will depend on the initial condition of the different clocks and the graph topology. In general we will require the graph $G(V, E)$ to be *connected*.

Definition 1: We say that the graph $G(V, E)$ is *connected* if there is at least one node $i \in V$ such that every other node j has a directed path to it.

IV. ANALYSIS

In this section we analyze the asymptotic behavior of system (12) and provide a necessary and sufficient condition on the parameter values that guarantees convergence to (13). The techniques used are drawn from the control community, e.g. [5] and [16], yet its application in our case is nontrivial.

Notation: We use $\mathbf{0}_{m \times n}$ ($\mathbf{1}_{m \times n}$) to denote the matrices of all zeros (ones) within $\mathbb{R}^{m \times n}$ and $\mathbf{0}_n$ ($\mathbf{1}_n$) to denote the column vectors of appropriate dimensions. $I_n \in \mathbb{R}^{n \times n}$ represents the identity matrix. Given a matrix $A \in \mathbb{R}^{n \times n}$ with Jordan normal form $A = PJP^{-1}$, let $n_A \leq n$ denote the total number of Jordan blocks J_l with $l \in \mathcal{I}(A) := \{1, \dots, n_A\}$. We use $\mu_l(A)$, $l \in \{1, \dots, n\}$ or just $\mu(A)$ to denote the eigenvalues of A , and order them decreasingly $|\mu_1(A)| \geq \dots \geq |\mu_n(A)|$. Finally, A^T is the transpose of A , A_{ij} is the element of the i th row and j th column of A and a_i is the i th element of the column vector a , i.e. $a = [a_i]^T$.

It is more convenient for the analysis to use a vector form representation of (12) given by

$$z_{k+1} = \Gamma_k z_k \quad (14)$$

where $z_k := [x(t_k)^T s(t_k)^T y(t_k)^T]^T$ and

$$\Gamma_k := \begin{bmatrix} I_n & \tau_k R & 0 \\ -\kappa_1 c L & I_n & -\kappa_2 I_n \\ p(-cL) & \mathbf{0}_{n \times n} & (1-p)I_n \end{bmatrix}$$

and L is the Laplacian matrix associated with $G(V, E)$,

$$L_{ii} = 1 \text{ and } L_{ij} = \begin{cases} -(d_i)^{-1} & \text{if } ij \in E, \\ 0 & \text{otherwise.} \end{cases}$$

The convergence analysis of this section is done in two stages. First, we provide necessary and sufficient conditions for synchronization in terms of the eigenvalues of Γ_k (Section IV-A) and then use Hermite-Biehler Theorem [17] to relate these eigenvalues with the parameter values (Section IV-B).

A. Asymptotic Behavior

We start by studying the asymptotic behavior of (14). That is, we are interested in finding under what conditions the series of $x_i(t_k)$ converge to (13).

We will assume WLOG that $\tau_k = \tau \forall k$ to simplify presentation. The proofs presented in this paper can be readily extended for the time varying τ_k . Thus, we will drop the k index of Γ_k from here on.

Consider the Jordan normal form [18] of Γ

$$\Gamma = [\zeta_1 \quad \dots \quad \zeta_{3n}] J [\eta_1 \quad \dots \quad \eta_{3n}]^T$$

where $J = \text{blockdiag}(J_l)_{l \in \mathcal{I}(\Gamma)}$, ζ_i and η_i are the right and left generalized eigenvectors of Γ such that

$$\zeta_i^T \eta_j = \begin{cases} 1 & \text{if } j = i, \\ 0 & \text{otherwise.} \end{cases}$$

The crux of the analysis comes from understanding the relationship between the multiplicity of the eigenvalue $\mu(\Gamma) = 1$

and the eigenvalue $\mu(L) = 0$, and their corresponding eigenvectors.

Lemma 1 (Eigenvalues of Γ and Multiplicity of $\mu(\Gamma) = 1$): Γ has an eigenvalue $\mu(\Gamma) = 1$ with multiplicity 2 if and only if the graph $G(V, E)$ is connected, $\kappa_1 \neq \kappa_2$ and $p > 0$.

Furthermore, $\mu(\Gamma)$ are the roots of

$$g_l(\lambda) := (\lambda - 1)^2(\lambda - 1 + p) + [(\lambda - 1)\kappa_1 + \kappa_2 - \kappa_1]\nu_l \quad (15)$$

where $\nu_l = \mu_l(\tau cLR)$ and satisfies

$$\nu_n = 0 < |\nu_l| \text{ for } l \in \{1, \dots, n-1\}. \quad (16)$$

Lemma 2 (Jordan Chains of $\mu(\Gamma) = 1$): Under the conditions of Lemma 1 the right and left Jordan chains, (ζ_1, ζ_2) and (η_2, η_1) respectively, associated with $\mu(\Gamma) = 1$ are given by

$$\zeta_1 = [\mathbf{1}_n^T \ \mathbf{0}_n^T \ \mathbf{0}_n^T]^T, \zeta_2 = [\mathbf{0}_n^T \ \frac{(R^{-1}\mathbf{1}_n)^T}{\tau} \ \mathbf{0}_n^T]^T, \quad (17)$$

$$\eta_2 = [\mathbf{0}_n^T \ \tau\alpha\xi^T \ \frac{-\tau\alpha\kappa_2}{p}\xi^T]^T, \eta_1 = [(\alpha R^{-1}\xi)^T \ \mathbf{0}_n^T \ \mathbf{0}_n^T]^T \quad (18)$$

where ξ is the unique left eigenvector of $\mu(L) = 0$ and α is the ξ_i -weighted harmonic mean of r_i , i.e.

$$\frac{1}{\alpha} = \mathbf{1}_n^T R^{-1} \xi = \sum_{i=1}^n \frac{\xi_i}{r_i}. \quad (19)$$

The proof of Lemmas 1 and 2 can be found in the Appendix. We now proceed to state and prove our first convergence result.

Theorem 1 (Convergence): The algorithm (12) achieves synchronization for any initial conditions if and only if the graph $G(V, E)$ is connected, $\kappa_1 \neq \kappa_2$, $p > 0$ and $|\mu_l(\Gamma)| < 1$ whenever $\mu_l(\Gamma) \neq 1$. Moreover, whenever the system synchronizes, we have

$$x^* = \alpha \sum_{i=1}^n \frac{\xi_i}{r_i} x_i(0), \quad r^* = \alpha \sum_{i=1}^n \xi_i (s_i(0) - \frac{\kappa_2}{p} y_i(0)). \quad (20)$$

Proof: We first notice that whenever $x(t_k)$ approaches (13) then

$$\lim_{h \rightarrow \infty} x(t_h) - r^* \mathbf{1}_n t_h = x^* \mathbf{1}_n \quad (21)$$

Sufficiency: Since we are under the assumptions of Lemmas 1 and 2 we know that $\mu(\Gamma) = 1$ has multiplicity 2 and a Jordan chain of size 2. Thus, the Jordan normal form of Γ is

$$J = [\zeta_1 \dots \zeta_{3n}] \begin{bmatrix} 1 & 1 & \mathbf{0}_{2 \times (3n-2)} \\ 0 & 1 & \\ \mathbf{0}_{(3n-2) \times 2} & \hat{J} & \end{bmatrix} \begin{bmatrix} \eta_1^T \\ \vdots \\ \eta_{3n}^T \end{bmatrix}$$

where \hat{J} has eigenvalues with $|\mu(\hat{\Gamma})| \leq \rho < 1$. Thus, it follows that

$$\begin{aligned} \lim_{h \rightarrow \infty} \Gamma^h - \zeta_1 \eta_1^T - (h\zeta_1 + \zeta_2) \eta_2^T &= \\ &= [\zeta_1 \dots \zeta_{3n}] \begin{bmatrix} \mathbf{0}_{2 \times 2} & \mathbf{0}_{2 \times (3n-2)} \\ \mathbf{0}_{(3n-2) \times 2} & \hat{J}^h \end{bmatrix} \begin{bmatrix} \eta_1^T \\ \vdots \\ \eta_{3n}^T \end{bmatrix} = 0 \end{aligned} \quad (22)$$

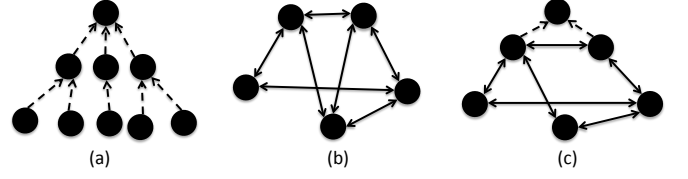


Fig. 5: Graphs with real eigenvalue Laplacians

where the last equality follows since $\rho < 1$, i.e.

$$\|\hat{J}^h\| \leq \|\hat{J}\|^h \leq \rho^h \xrightarrow{h \rightarrow \infty} 0.$$

Now, using (17) and (18) we get

$$\zeta_1 \eta_1^T = \begin{bmatrix} \alpha \mathbf{1}_n^T R^{-1} & \mathbf{0}_{n \times n} & \mathbf{0}_{n \times n} \\ \mathbf{0}_{n \times n} & \mathbf{0}_{n \times n} & \mathbf{0}_{n \times n} \\ \mathbf{0}_{n \times n} & \mathbf{0}_{n \times n} & \mathbf{0}_{n \times n} \end{bmatrix}$$

$$(h\zeta_1 + \zeta_2) \eta_2^T = \begin{bmatrix} \mathbf{0}_{n \times n} & h\tau\alpha \mathbf{1}_n \xi^T & -h\tau\alpha \frac{\kappa_2}{p} \mathbf{1}_n \xi^T \\ \mathbf{0}_{n \times n} & \alpha R^{-1} \mathbf{1}_n \xi^T & -\alpha \frac{\kappa_2}{p} R^{-1} \mathbf{1}_n \xi^T \\ \mathbf{0}_{n \times n} & \mathbf{0}_{n \times n} & \mathbf{0}_{n \times n} \end{bmatrix}$$

Therefore, given any initial condition $z_0 = (x_0^T, s_0^T, y_0^T)^T$, we have

$$\lim_{h \rightarrow \infty} x(t_h) - t_h \alpha \mathbf{1}_n \xi^T (s_0 - \frac{\kappa_2}{p} y_0) = \alpha \mathbf{1}_n \xi^T R^{-1} x_0 \quad (23)$$

and (20) follows from identifying (23) and (21).

Necessity: The algorithm achieves synchronization whenever (21) holds. Suppose by contradiction, that one of the conditions of the theorem does not hold. Then by Lemmas 1 and 2 there is at least another Jordan block with $|\mu_{l_0}(\Gamma)| \geq 1$. We have two cases: 1) If $|\mu_{l_0}(\Gamma)| > 1$, then the limits (22) and (21) do not exist (contradiction). 2) If $|\mu_{l_0}(\Gamma)| = 1$, then the limit (22) might exist. However, since $\xi_{l_0}^T \xi_1 = 0$ and $\xi_{l_0}^T \xi_2 = 0$ the asymptotic contribution to $x(t_h)$, $s(t_h)$ or $y(t_h)$ will be non-homogeneous among different members. Thus, (21) does not hold (contradiction). ■

B. Necessary and sufficient conditions of synchronization

We now provide necessary and sufficient conditions on the parameter values for Theorem 1 to hold. We will restrict our attention to graphs that have Laplacian matrices with real eigenvalues. This includes for example trees, symmetric graphs ($ij \in E$ iff $ji \in E$) and symmetric graphs with a root. See Figure 5.

The proof consists on studying the Schur stability of $g_l(\lambda)$ and has several steps. We first perform a change of variable that maps the unit circle onto the left half-plane. This transforms the problem of studying the Schur stability into a Hurwitz stability problem which is solved using Hermite-Beihler Theorem.

Theorem 2 (Hurwitz Stability (Hermite-Beihler)): Given the polynomial $P(s) = a_n s^n + \dots + a_0$, let $P^r(\omega)$ and $P^i(\omega)$ be the real and imaginary part of $P(j\omega)$, i.e. $P(j\omega) = P^r(\omega) + jP^i(\omega)$. Then $P(s)$ is a Hurwitz polynomial if and only if

- 1) $a_n a_{n-1} > 0$
- 2) The zeros of $P^r(\omega)$ and $P^i(\omega)$ are all simple and real and interlace as ω runs from $-\infty$ to $+\infty$.

Proof: See [17]. ■

We now compute the parameter values.

Theorem 3 (Parameter Values for Synchronization): Given a connected graph $G(V, E)$ such that L has real eigenvalues. The algorithm (12) achieves synchronization if and only if

- (i) $|1 - p| < 1$ or equivalently $2 > p > 0$
- (ii) $\frac{2\kappa_1}{3p} > \kappa_1 - \kappa_2 > 0$
- (iii) $\tau < \frac{p(\kappa_2 - p(\kappa_1 - \kappa_2))}{c\mu_{\max}(\kappa_1 - p(\kappa_1 - \kappa_2))^2}$

where μ_{\max} is the largest eigenvalue of LR .

Proof: We will show that when $G(V, E)$ is connected with $\mu(L) \in \mathbb{R}$, then (i)-(iii) are equivalent to the conditions of Theorem 1.

Since, $G(V, E)$ is connected and (i)-(ii) satisfies $p > 0$ and $\kappa_1 \neq \kappa_2$, the conditions of Lemma 1 are satisfied. Therefore the multiplicity of $\mu(\Gamma) = 1$ is two and by (16) these are the roots of

$$g_n(\lambda) = (\lambda - 1)^2(\lambda - 1 + p),$$

which corresponds to the case $\nu_n = 0$.

Thus, to satisfy Theorem 1 we need to show that the remaining eigenvalues are strictly in the unit circle. This is true for the remaining root of $g_n(\lambda)$ iff (i).

For the remaining $g_l(\lambda)$, this implies that are Schur polynomials. Thus, we will show that $g_l(\lambda)$ is a Schur polynomial if and only if (i)-(iii) hold. We drop the subindex l for the rest of the proof.

We first transform the Schur stability problem into a Hurwitz stability problem. Consider the change of variable $\lambda = \frac{s+1}{s-1}$. Then $|\lambda| < 1$ if and only if $\mathbb{R}[s] < 0$.

Now, since $\nu > 0$ by (16), let

$$P(s) = \frac{(s-1)^3}{\delta\kappa p\nu} g\left(\frac{s+1}{s-1}\right) = s^3 + \left(\frac{2\kappa_1}{\delta\kappa p} - 3\right)s^2 + \left(\frac{4}{\delta\kappa\nu} + 3 - \frac{4\kappa_1}{\delta\kappa p}\right)s + \frac{4(2-p)}{\delta\kappa p\nu} + \frac{2\kappa_1}{\delta\kappa p} - 1$$

where $\delta\kappa = \kappa_1 - \kappa_2$.

We will apply Hermite-Beihler Theorem to $P(s)$, but first let us express what 1) and 2) of Theorem (2) means here.

Condition 1) becomes

$$\frac{2\kappa_1}{\delta\kappa p} - 3 > 0. \quad (24)$$

Now let $P^r(\omega)$ and $P^i(\omega)$ be as in Theorem 2, i.e. let

$$P^r(\omega) = -\left(\frac{2\kappa_1}{\delta\kappa p} - 3\right)\omega^2 + \frac{4(2-p)}{\delta\kappa p\nu} + \frac{2\kappa_1}{\delta\kappa p} - 1$$

$$P^i(\omega) = -\omega^3 + \left(\frac{4}{\delta\kappa\nu} + 3 - \frac{4\kappa_1}{\delta\kappa p}\right)\omega$$

The roots of $P^r(\omega)$ are given by $\omega_0 = \pm\sqrt{\omega_0^r}$ where

$$\omega_0^r := \frac{\frac{4(2-p)}{\delta\kappa p\nu} + \frac{2\kappa_1}{\delta\kappa p} - 1}{\frac{2\kappa_1}{\delta\kappa p} - 3}, \quad (25)$$

and the roots of $P^i(\omega)$ by $\omega_0 \in \{0, \pm\sqrt{\omega_0^i}\}$ where

$$\omega_0^i := \frac{4}{\delta\kappa\nu} + 3 - \frac{4\kappa_1}{\delta\kappa p} \quad (26)$$

Since the roots $P^r(\omega)$ and $P^i(\omega)$ must be real, we must have $\omega_0^r > 0$ and $\omega_0^i > 0$. Therefore, by monotonicity of the square root, the interlacing condition 2) is equivalent to

$$0 < \omega_0^r < \omega_0^i. \quad (27)$$

Thus we will show: (i)-(iii) hold \iff (24) and (27) hold.

It is straightforward to see that using (i) and (ii) we can get (24). On the other hand, $\omega_0^i > 0$ from (27) together with (24) gives $0 < \frac{4}{\delta\kappa\nu} + 3 - \frac{4\kappa_1}{\delta\kappa p} < \frac{4}{\delta\kappa\nu}$, which implies that $\delta\kappa > 0$, and therefore (ii) follows.

Now using (24) and (25), $\omega_0^r > 0$ becomes

$$\frac{4(2-p)}{\delta\kappa p\nu} + \frac{2\kappa_1}{\delta\kappa p} - 1 > 0$$

which always holds under (i) and (ii) since the first term is always positive and $\frac{2\kappa_1}{\delta\kappa p} - 1 > \frac{2\kappa_1}{\delta\kappa p} - 3 > 0$ by (24).

Finally, using (25) and (26), $\omega_0^r < \omega_0^i$ becomes

$$\frac{\frac{4(2-p)}{\delta\kappa p\nu} + \frac{2\kappa_1}{\delta\kappa p} - 1}{\frac{2\kappa_1}{\delta\kappa p} - 3} < \frac{4}{\delta\kappa\nu} + 3 - \frac{4\kappa_1}{\delta\kappa p}$$

$$\frac{\frac{2\kappa_1}{\delta\kappa p} - 1}{\frac{2\kappa_1}{\delta\kappa p} - 3} + \frac{4\kappa_1}{\delta\kappa p} - 3 < \frac{4}{\delta\kappa\nu} \left(1 - \frac{(2-p)}{\frac{2\kappa_1}{\delta\kappa p} - 3}\right)$$

where the left-hand side (LHS) is

$$LHS = \frac{(2\kappa_1 - \delta\kappa p)\delta\kappa p + (4\kappa_1 - 3\delta\kappa p)(2\kappa_1 - 3\delta\kappa p)}{(2\kappa_1 - 3\delta\kappa p)\delta\kappa p}$$

$$= \frac{8(\kappa_1^2 - 2\kappa_1\delta\kappa p + (\delta\kappa p)^2)}{(2\kappa_1 - 3\delta\kappa p)\delta\kappa p} = \frac{8(\kappa_1 - \delta\kappa p)^2}{(2\kappa_1 - 3\delta\kappa p)\delta\kappa p}$$

and the right hand side (RHS) is

$$RHS = \frac{4}{\delta\kappa\nu} \frac{\frac{2\kappa_1 - 3\delta\kappa p + (2-p)\delta\kappa}{\delta\kappa p}}{\frac{2\kappa_1 - 3\delta\kappa p}{\delta\kappa p}} = \frac{8}{\delta\kappa\nu} \frac{\kappa_2 - \delta\kappa p}{2\kappa_1 - 3\delta\kappa p}.$$

Thus $LHS < RHS$ becomes

$$\frac{8(\kappa_1 - \delta\kappa p)^2}{(2\kappa_1 - 3\delta\kappa p)\delta\kappa p} < \frac{8}{\delta\kappa\nu} \frac{\kappa_2 - \delta\kappa p}{2\kappa_1 - 3\delta\kappa p}$$

$$\frac{(\kappa_1 - \delta\kappa p)^2}{p} < \frac{1}{\nu}(\kappa_2 - \delta\kappa p)$$

$$\nu < \frac{p(\kappa_2 - \delta\kappa p)}{(\kappa_1 - \delta\kappa p)^2} \quad (28)$$

Finally, $\nu_l = \mu_l(\tau cLR) = \tau c\mu_l(LR)$. Thus, since (28) should hold $\forall l \in \{1, \dots, n-1\}$, then

$$\tau < \min_l \frac{p(\kappa_2 - \delta\kappa p)}{c\mu_l(LR)(\kappa_1 - \delta\kappa p)^2} = \frac{p(\kappa_2 - \delta\kappa p)}{c\mu_{\max}(\kappa_1 - \delta\kappa p)^2}$$

which is exactly (iii). ■

Even though μ_{\max} depends on r_i which is in general unknown, it is easy to show that $\mu_l(LR) \leq \hat{r}_{\max}\mu_l(L)$ where \hat{r}_{\max} is an upper bound of the maximum rate deviation r_i .

Furthermore, using Greshgorin's circle theorem, it is easy to show that $\mu_{\max}(L) \leq 2$. Therefore, if we set

$$\tau < \frac{p(\kappa_2 - \delta\kappa p)}{\hat{2}r_{\max}(\kappa_1 - \delta\kappa p)^2} \quad (29)$$

convergence is guaranteed for every graph with real eigenvalues.

V. EXPERIMENTS

We implement an asynchronous version of our algorithm in C using the IBM CCT solution as our code base. Our program reads the TSC counter directly using the `rdtsc` assembly instruction to minimize reading latencies and maintains a virtual clock that can be directly updated. The list of neighbors is read from a configuration file and whenever there is no neighbor, the program follows the local Linux clock. Finally, offset measurements are taken using an improved ping pong mechanism proposed in [7].

We run our skewless protocol in a cluster of IBM BladeCenter LS21 servers with two AMD Opteron processors of 2.40GHz, and 16GB of memory. As shown in Figure 2, the servers `serv1-serv10` are used to run the protocol. The offset measurements are taken through a Gigabit Ethernet switch. Server `serv0` is used as a reference node and gathers time information from the different nodes using a Cisco 4x InfiniBand Switch that supports up to 10Gbps between any two ports and up to 240Gbps of aggregate bandwidth. This minimizes the error induced by the data collecting process.

We use this testbed to validate the analysis in Section IV. First, we illustrate the effect of different parameters and analyze the effect of the network configuration on convergence. (Experiment 1) Then we present a series of configurations that demonstrate how connectivity between clients is useful in reducing the jitter of a noisy clock source. (Experiment 2) And finally, we compare the performance of the algorithm with respect to the skew based algorithm proposed in [7] (Experiment 3).

Unless explicitly stated, the default parameter values used are

$$p = 0.99, \quad \kappa_1 = 1.1 \text{ and } \kappa_2 = 1.0. \quad (30)$$

Even though Theorem 3 allows p to have any value between 0 and 2 the behavior of the algorithm is smoother when $p \in (0, 1)$, i.e. when $y_i(t_k)$ is in fact a weighted average.

Notice that these values immediately satisfy (i) and (ii) of Theorem 3 since $1-p = 0.01$, $\frac{2\kappa_1}{3p} = 0.7407 > \kappa_1 - \kappa_2 = 0.1$. The remaining condition can be satisfied by modifying τ or equivalently c . Here, we choose to fix $c = 0.7$ which makes condition (iii)

$$\tau < \frac{1.2717}{\mu_{\max}} \text{ s.}$$

For fixed time step τ , the stability of the system depends on the value of μ_{\max} .

Experiment 1: We first consider the client server configuration described in Figure 6 (a) with a time step

$$\tau = 1 \text{ s.} \quad (31)$$

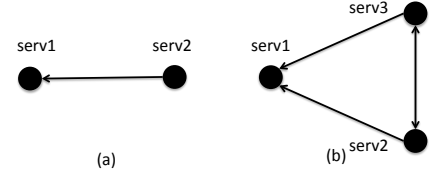
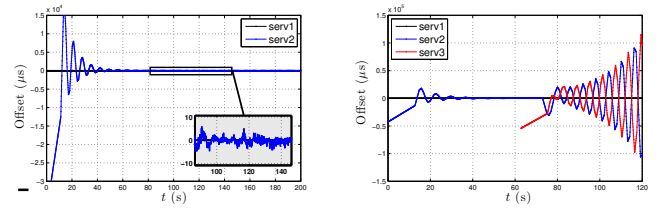


Fig. 6: Effect of topology on convergence: (a) client-server configuration (b) Two clients connected to server and mutually connected

In this configuration $\mu_{\max} = 1$ and therefore condition (iii) becomes $\tau < 1.2717 \text{ s}$. Figure 7a shows the offset between `serv1` (the leader) and `serv2` (the client) in microseconds. There we can see how `serv2` gradually updates s_2 until the offset becomes insignificant.



(a) Client server configuration: $\tau = 1 \text{ s}$ (b) Two clients mutually connected with $\tau = 1 \text{ s}$

Fig. 7: Offset of the nine servers connected to a noisy clock source

Figure 7a tends to suggest that the set of parameters given by (30) and (31) are suitable for deployment on the servers. This is in fact true provided that network is a directed tree as in Figure 5 (a). The intuition behind this fact is that in a tree, each client connects only to one server. Thus, those connected to the leader will synchronize first and then subsequent layers will follow.

However, once loops appear in the network there is no longer a clear dependency since two given nodes can mutually get information from each other. This type of dependency might make the algorithm unstable.

Figure 7b shows an experiment with the same configuration as Figure 7a in which `serv2` synchronizes with `serv1` until a third server (`serv3`) appears after 60s. At that moment the system is reconfigured to have the topology of Figure 6 (b) introducing a timing loop between `serv2` and `serv3`. This timing loop makes the system unstable.

The instability arises since after `serv3` starts the topology changes, setting $\mu_{\max} = 1.5$. Thus, the time step condition (iii) becomes $\tau < 847.8 \text{ ms}$ which is no longer satisfied by $\tau = 1 \text{ s}$.

Using (29) we can recover the stability of the system by setting

$$\tau = 500 \text{ ms} < \frac{1.2717}{2} \text{ s} = 645.85 \text{ ms}$$

Figure 8 shows how now `serv2` and `serv3` can synchronize with `serv1` after introducing this change.

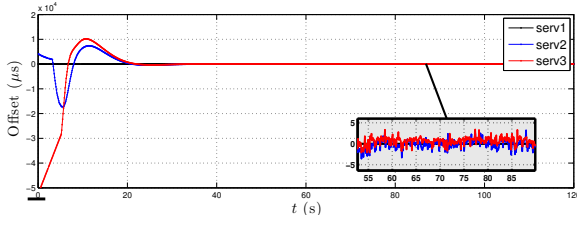


Fig. 8: Two clients mutually connected with $\tau = 500\text{ms}$

Experiment 2: We now show how timing loops can be used to collectively outperform individual clients when the time source is noisy. As performance metric we choose the *mean relative deviation* from the leader, i.e. $\sqrt{S_n}$ where

$$S_n = \frac{1}{n-1} \sum_{i=2}^n E(x_i - x_1)^2. \quad (32)$$

We disable the offset filter to observe the worst case scenario.

We run our algorithm on 10 servers (serv1 through serv10). The connection setup is described in Figure 9. Every node is directly connected unidirectionally to the leader (serv1) and bidirectionally to $2K$ additional neighbors. When $K = 0$ then

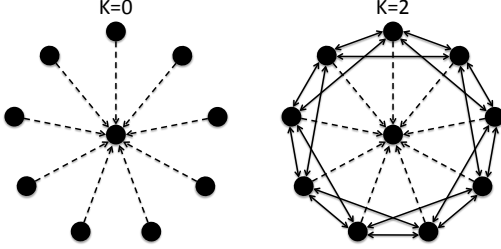


Fig. 9: Leader topologies with $2K$ neighbors connection. Connections to the leader (serv1) are unidirectional while the connections among clients (serv2 through serv10) are bidirectional

the network reduces to a star topology and when $K = 4$ the servers serv2 through serv10 form a complete graph.

The dashed arrows in Figure 9 show the connections where jitter was introduced. To emulate a link with jitter we added random noise η with values taken uniformly from $\{0, 1, \dots, \text{Jitter}_{\max}\}$ on both direction of the communication.

$$\eta \in \{0, 1, \dots, \text{Jitter}_{\max}\} \text{ms} \quad (33)$$

Notice that the arrow only shows a dependency relationship, the ping pong mechanism sends packets in both direction of the physical communication. We used a value of $\text{Jitter}_{\max} = 10\text{ms}$. Since the error was introduced in both directions of the ping pong, this is equivalent to a standard deviation of 6.05ms .

Figure 10 illustrates the relative offset between the two extreme cases; The star topology ($K = 0$) is shown in Figure 10a, and the complete subgraph ($K = 4$) is shown in Figure 10b.

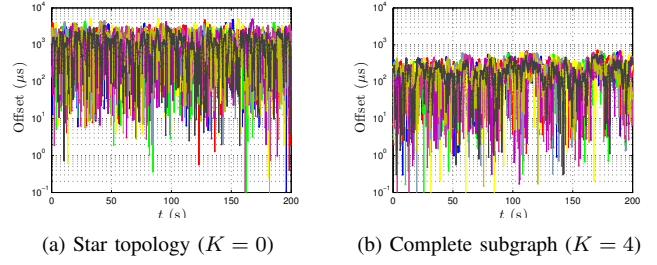


Fig. 10: Offset of the nine servers connected to a noisy clock source

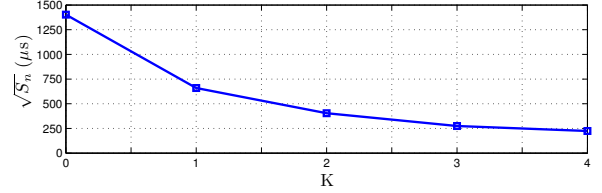


Fig. 11: Effect of the client's communication topology on the mean relative deviation. As the connectivity increases (K increases) the mean relative deviation is reduced by factor of 6.26, i.e. a noise reduction of approx. 8dB.

The worst case offset for $K = 0$ is 5.1ms which is on the order of the standard deviation of the jitter. However, when $K = 4$ we obtain a worst case offset of $690.8\mu\text{s}$.

The mean relative deviation $\sqrt{S_n}$ as the connectivity among clients increases from isolated nodes ($K = 0$) to a complete subgraph ($K = 4$) is studied in Figure 11. The results presented show that without any type of error filtering the network itself is able to perform a distributed filtering that achieves an improvement of up to a factor of 6.26 or equivalently a noise reduction of almost 8dB.

Experiment 3: We now proceed to compare the performance of our algorithm (alg1 in the figures) with respect to the one proposed in [7] (alg2). Since alg2 performs some filtering, for the purpose of fairness we filter the offset measurements of alg1 by only accepting those whose round trip time are within twice the mean deviation. We use $c = 0.35$ and $\tau = 250\text{ms}$.

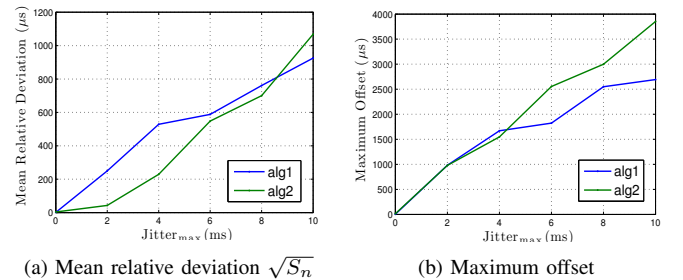


Fig. 12: Performance evaluation between our solution (alg1) and IBM CCT (alg2)

In Figure 12a we present the mean relative deviation $\sqrt{S_n}$ for a star topology as the jitter is increased from $\text{Jitter}_{\max} = 0\text{ms}$ (no jitter) to $\text{Jitter}_{\max} = 10\text{ms}$ with a granularity of 1ms . The worst case offset is shown in Figure 12b. While the mean relative deviation is slightly better for alg2, the maximum offset is better for our algorithm when the jitter is high. This is because extreme burst of jitter can introduce large errors in the estimation of the skew which makes alg2 steer significantly.

VI. CONCLUSION

This paper presents a clock synchronization protocol that is able to synchronize networked nodes without explicit estimation of the clock skews and steep corrections on the time. Unlike current standards, our protocol is guaranteed to converge even in the presence of timing loops which allow different clients to share timing information and collectively outperform individual clients when the time source has large jitter. We implemented our solution on a cluster of IBM BladeCenter servers and report its performance.

REFERENCES

- [1] D. Mills, "Network time protocol version 4 reference and implementation guide," University of Delaware, Tech. Rep. 06-06-1, June 2006.
- [2] "Ieee standard for a precision clock synchronization protocol for networked measurement and control systems," pp. 1–269, 2008.
- [3] A. Sobeih, M. Hack, Z. Liu, and null Li Zhang, "Almost peer-to-peer clock synchronization," *Parallel and Distributed Processing Symposium, International*, vol. 0, p. 21, 2007.
- [4] D. Veitch, J. Ridoux, and S. B. Korada, "Robust synchronization of absolute and difference clocks over networks," *IEEE/ACM Trans. Netw.*, vol. 17, no. 2, pp. 417–430, Apr. 2009. [Online]. Available: <http://dx.doi.org/10.1109/TNET.2008.926505>
- [5] R. Carli and S. Zampieri, "Networked clock synchronization based on second order linear consensus algorithms," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, dec. 2010, pp. 7259–7264.
- [6] E. Mallada and A. Tang, "Distributed clock synchronization: Joint frequency and phase consensus," in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, dec. 2011, pp. 6742–6747.
- [7] S. Froehlich, M. Hack, X. Meng, and L. Zhang, "Achieving precise coordinated cluster time in a cluster environment," in *Precision Clock Synchronization for Measurement, Control and Communication, 2008. ISPCS 2008. IEEE International Symposium on*, September 2008, pp. 54–58.
- [8] L. Zhang, Z. Liu, and C. Honghui Xia, "Clock synchronization algorithms for network measurements," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, 2002, pp. 160–169 vol.1.
- [9] H. Kim, X. Ma, and B. Hamilton, "Tracking low-precision clocks with time-varying drifts using kalman filtering," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 1, pp. 257–270, feb. 2012.
- [10] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, 2002.
- [11] D. Hunt, G. Korniss, and B. Szymanski, "Network synchronization in a noisy environment with time delays: Fundamental limits and trade-offs," *Physical Review Letters*, vol. 105, no. 6, p. 068701, 2010.
- [12] H. Marouani and M. R. Dagenais, "Internal clock drift estimation in computer clusters," *J. Comp. Sys., Netw., and Comm.*, vol. 2008, pp. 9:1–9:7, Jan. 2008. [Online]. Available: <http://dx.doi.org/10.1155/2008/583162>
- [13] S. Moon, P. Skelly, and D. Towsley, "Estimation and removal of clock skew from network delay measurements," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, mar 1999, pp. 227–234 vol.1.
- [14] M. Lemmon, J. Ganguly, and L. Xia, "Model-based clock synchronization in networks with drifting clocks," in *Dependable Computing, 2000. Proceedings. 2000 Pacific Rim International Symposium on*, 2000, pp. 177–184.
- [15] D. Mills, "Network time protocol (version 3) specification, implementation and analysis," 1992.
- [16] D. Xie and S. Wang, "Consensus of second-order discrete-time multi-agent systems with fixed topology," *Journal of Mathematical Analysis and Applications*, 2011.
- [17] S. Bhattacharyya, H. Chapellat, and L. Keel, *Robust control*. Prentice-Hall Upper Saddle River, New Jersey, 1995.
- [18] R. Horn and C. Johnson, *Matrix analysis*. Cambridge Univ Pr, 1990.

APPENDIX

A. Proof of Lemma 1

Proof: We first compute the characteristic polynomial of Γ . In order to compute

$$\det(\lambda I_{3n} - \Gamma) = \begin{vmatrix} (\lambda - 1)I_n & -\tau R & 0 \\ \kappa_1 cL & (\lambda - 1)I_n & \kappa_2 I_n \\ pcL & 0 & (\lambda - 1 + p)I_n \end{vmatrix}$$

we will iteratively use the following property of the determinant of block matrices $\det(A) = \det(A_{11}) \det(A \setminus A_{11})$ where $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ and $A \setminus A_{11} = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is the Schur complement of A_{11} [18].

Thus, it follows

$$\begin{aligned} \det(\lambda I_{3n} - \Gamma) &= \det((\lambda - 1)I_n) \det((\lambda I_{3n} - \Gamma) \setminus (\lambda - 1)I_n) \\ &= (\lambda - 1)^n \begin{vmatrix} (\lambda - 1)I_n + \frac{\tau \kappa_1}{\lambda - 1} cLR & \kappa_2 I_n \\ \frac{\tau p}{\lambda - 1} cLR & (\lambda - 1 + p)I_n \end{vmatrix} \\ &= \det((\lambda - 1)^2(\lambda - 1 + p)I_n + [(\lambda - 1)\kappa_1 \\ &\quad + (\kappa_2 - \kappa_1)]\tau cLR) = \prod_{l=1}^n g_l(\lambda), \end{aligned}$$

where $g_l(\lambda)$ is as defined in (15).

Thus, $\lambda = 1$ is a double root of the characteristic polynomial if and only if $\kappa_1 \neq \kappa_2$, $p > 0$ and τcLR has a simple zero eigenvalue, i.e. (16). Now, since R is nonsingular (16) must hold for the eigenvalues of L as well, which is in fact true if and only if the directed graph $G(V, E)$ is connected [16]. ■

B. Proof of Lemma 2

Proof: We start by computing the right Jordan chain. By definition of ζ_1 , $(\Gamma - I)\zeta_1 = 0_n$. Thus, if $\zeta_1 = [x^T \ s^T \ y^T]^T$, then the following system of equations must be satisfied

$$\tau R s = 0 \text{ (a)}, \quad -\kappa_1 cLx - \kappa_2 y = 0 \text{ (b)}, \quad -pcLx - py = 0 \text{ (c)}. \quad (34)$$

Equation (34a) implies $s = 0$. Now, (34c) implies $cLx = -y$, which when substituted in (34b) gives $(\kappa_2 - \kappa_1)y = 0$. Thus, since $\kappa_1 \neq \kappa_2$, $y = 0$ and $x \in \ker(L)$. By choosing $x = \mathbf{1}_n$ we obtain ζ_1 . Notice that the computation also shows that ζ_1 is the unique eigenvector of $\mu(\Gamma) = 1$ which implies that there is only one Jordan block of size 2. The second member of the chain, ζ_2 , can be computed similarly by solving $(\Gamma - I)\zeta_2 = \zeta_1$.

The vectors η_2 and η_1 can be solved in the same way using $\eta_2^T(\Gamma - I) = 0$ and $\eta_1^T(\Gamma - I) = \eta_2^T$ which gives (18). Equation (19) follows from imposing $\zeta_1^T \eta_1 = 1$. ■